A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

LA-UR -85-2838

CONF

CONF-8509149--1

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

LA-UR--85-2838

DE85 017565

TITLE Expert Systems for Design and Simulation

AUTHOR(S) Jack Aldridge, John Cerutti, Willard Draisin, and Michael Steuerwalt

SUBMITTED TO AIAA/NASA Symposium on Automation, Robotics and Advanced Computing for the National Space Program

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

EXPERT SYSTEMS FOR DESIGN AND SIMULATION

Jack Aldridge,
John Cerutti,
Willard Draisin,
and
Michael Steuerwalt

Los Alamos National Laboratory
Los Alamos, NM 87545

Abstract

We discuss work in progress on two expert systems. We are developing systems that use artificial intelligence techniques to simplify the use of large simulation codes and to help design complicated physical devices. The simulation codes are used in analyzing and designing weapons, and the devices are themselves parts of weapon systems. But we focus not only on the particular applications, but also on the broader issues common to design problems: large solution spaces and tentative reasoning. We also discuss some practical difficulties encountered during the project. One expert system provides an interface between users and several simulation codes. It checks input for errors, builds input files for the codes, and submits jobs to a central computing facility. The other expert system helps turn a description of a device into a particular design. Currently this expert system includes three major parts: a translator of descriptions into designs, a graphics interface that presents the design to the user and allows him to manipulate it, and a refiner of designs. The latter is the "smartest" part of the system, and the target of much of our present efforts.

Introduction

The major task of the Los Alamos National Laboratory is the design of nuclear weapons. This design process requires the use of large FORTRAN simulation codes that run on the fastest supercomputers available. To a

large extent, design is an art that consumes great time of both people and computers. Therefore we are constructing two expert systems to simplify the use of the simulation codes and to assist in the design of weapons. Both systems described here are works in progress. This paper should be viewed as an interim report.

Out of synch with their documentation, needing input in particular forms that are hard to remember, crashing for no discernible reason, big codes are often hard to use. But they are necessary: they exploit the capabilities of computers to do calculations that are otherwise impossible, and so increase our understanding and guide our intuition about complicated phenomena. However, using computers raises problems. We must assimilate information we already have, choose what to compute, describe the computation to the machine, interpret the machine's results, and combine the new information with what we already know. Fig. 1 shows the environment in which users now run simulation codes at Los Alamos. In these circumstances, the user bears the burden of understanding not only the relevant physics, say, of his problem, but also the inner details of the simulation code and the computing system. Both expert systems that we describe here address a common long-range goal: making the computer a more adaptable, expressive, amiable tool. Fig. 2 indicates the environment we aim for. Here inner details of simulation codes are handled by the controller expert system, while the apprentice expert system helps the user express a simulation in terms natural to him rather than to the underlying computational device. The apprentice also helps refine the design, set up a simulation, and interpret results.

Procon

Procon is an expert system that provides an interface between users and several simulation codes; it helps the designer set up and submit "production" calculations. We run simulation codes interactively when the problem is small enough. But large problems -- including most designs -- require several hours of Cray time, and so are run in "production" mode. A production controller is a control program containing both input parameters for a

simulation code and procedures to be followed by the production supervisor program to complete the computation or to preserve it should unforeseen difficulties arise. Currently designers rely on three such production controllers, each maintained by a different person. Procon will replace these controllers, thus providing a uniform user interface to the three codes and simplifying maintenance.

The expertise of Procon lies in its knowledge of what inputs are required for the weapons codes, what ranges and forms those parameters have, and what procedures are followed by the production supervisor. This information is gathered in databases that are manipulated by simple general control routines, a schema followed by many well-known expert systems.

Procon works in three major phases:

(1) Interface: it creates an environment for interaction with the user.

(2) Checker: it helps the user build input that describes the simulation to the weapons code.

(3) Generator: it constructs a job control stream and submits it for production.

The environment for the interaction includes such factors as the user's name, accounting information, last program used, and last data file used. More important are dynamic aspects, such as the disposition of output and a choice of actions the system should take if something goes awry. With this information, Procon allows a user who is working on an extensive design problem to continue his efforts without concern for these details, or to start work by perturbing his latest run. Procon helps describe the simulation by checking the form and range of input data for the simulation code. Such early detection of errors prevents the simulation code from crashing in a production run. Finally, generation of a job control stream and submittal are algorithmic tasks that ordinary controllers perform. Many of the same concepts carry over to Procon.

For users, Procon has two advantages over standard controllers: it provides a common interface to several

simulation codes, and it detects input errors in parameter names, values, subscripts, and formats. Of equal importance to us is an improvement that is unseen by users. The Procon program uses a variety of information about the inputs and expected behaviors of the simulation codes. These data are stored in database-like files separate from the executable program itself. In principle, then, we could add more target codes to Procon simply by extending the database, rather than by revising Procon's executable code. We have done this. Procon originally controlled a single simulation code. Exploiting the data-driven implementation of the checker, we added a second target code by augmenting the database, and are adding others.

Just as the checker is implemented with data-driven programming, so should the generator be. For instance, different simulation codes expect data in different formats. Moreover, big simulation codes last much longer than big computers. During the life of the codes details of production control procedures will change several times. A generator written with data-driven techniques need not be rewritten to track these changes. Although Procon's generator is not yet data-driven, we expect to make it so.

Friendly users received the first version of Procon on a Vax in July. During the next year we plan to make Procon operate in the production environment, to have Cray-Vax communication, to add a third weapons code, and to enhance Procon's error-checking abilities. Now single parameters are checked for errors; we are exploring ways to detect and correct more subtle errors, such as inconsistencies between parameters or violation of constraints imposed by the context or order of parameters.

Difficulties

Obviously, security constraints limit our choices of software, hardware, and ways to connect them. Economic factors also bind: giving $100,000 AI machines to every designer is not a practical possibility. Most of our users work on Cray computers. But there is no Lisp environment on the Crays to support AI development. For developing our systems we chose a Vax 780 running

EUNICE under VMS to emulate UNIX. We began with Gordon Novak's GLisp, exploiting its object-oriented programming features for database management. Because our Computing Division was putting Portable Standard Lisp (PSL) on the Cray, we also favored GLisp for its compatibility with PSL. But the PSL Flavors is considerably more primitive than the MIT Lisp Machine version available with Franz Lisp. We actually made a PSL version of Procon for the Cray. However, this program has not run because Cray PSL was not then a full implementation. Now we use Franz Lisp with the MIT macro enhancements.

One major difficulty is that EUNICE does not support complete input-output functions of UNIX. This disrupts the behavior of window systems, such as the University of Maryland window system, that we had planned to use. Lack of a window facility severely restricts the convenience of user interaction.

Another difficulty arises because our intended audience works on Crays and our system runs on a Vax. Designers are conservative. They prefer tools that they know work to tools that might be better but also might fail. That a Vax may have a richer and more supportive software environment than a Cray doesn't outweigh the inconvenience of learning a new operating system. They don't want to learn new operating systems; reasonably, they prefer to learn new physics. Consequently we need to provide access to Procon that doesn't make designers feel like they are in alien territory. Until Procon runs on the Crays, access means that the designer on a Cray can invoke Procon on a Vax without realizing that Procon isn't on the same machine where he is. This requires reliable and fast network communications. At present, Cray-Vax communication through the Integrated Computer Network at Los Alamos is not fast or dependable enough for this application to suit the designers.

The designer's apprentice

Our second expert system, a sort of designer's apprentice, helps turn a description of a weapon into a particular design. The apprentice is a direct attack on the problem of integrating numeric calculation with

symbolic computation. Our goal is to permit the designer to describe a design in terms natural to him. The apprentice should convert the description into a design, help him refine the design, recall related design problems, invoke standard simulation codes, and assist with parameter studies and interpretation of results. More precisely, the apprentice is to meet the following goals.

(1) It will propose a design to meet a requirement based on functional need. The requirement may be incompletely specified and may impose implicit constraints that the designer may not care to specify. The apprentice may propose a design by recalling a previous design, perturbing a prior design, or developing a new design based on heuristics and simple calculations.

(2) The apprentice will allow users to communicate with sophisticated simulation codes using language appropriate to the design discipline, rather than the syntax of a specific simulation code.

(3) It will provide a consistent interface to several simulation codes.

(4) The apprentice will embody a corporate memory both for designs and for design techniques. We believe that the corporate memory is necessary if the apprentice is to learn how to design better or if it is to help teach novice designers.

(5) It will help interpret the results of large-scale simulations. This includes both winnowing for the desired output and interpreting the output of one code for constructing input to another.

We are writing code that implements these objectives. Here we discuss the schema upon which this coding is based.

The apprentice is a software system rather than an isolated AI code; therefore it has utility features to make the job easier for the designer, as well as AI sections that attempt to capture expertise. Fig. 3 illustrates schematically the essential parts of the apprentice.

Input for the design is handled by a menu-like window that allows the designer to impose whatever

constraints he wants in whatever order he wants. When he indicates that he is finished, the program proposes a design based on designs that are known to work from testing or from model calculations, including heuristics, that can be done without the use of the powerful numerical simulators. (However, many of the heuristics are derived from considerable experience with the large simulators.) The space of possible designs is big. Because several design choices may be likely at a given stage, heuristics play an important role in choosing an alternative.

An early requirement was that the interaction with the designer be graphical, like CAD/CAM techniques. The apprentice presents its design to the user as a screen picture. The drawing is dynamic: the designer can point to a section and obtain information about it such as size, material, mass, and material properties via a pop-up window. Through the graphic interaction, he can add a new section to the design and add or change specific data. We use object-oriented programming for this. Material sections are objects whose attributes are parameters like those described above.

Perhaps the most challenging problem offered by the apprentice is the combination of inference with numerical simulation. The chief inferences we wish to make are that all constraints of the design are met or that no design that meets all constraints is possible. A trivial example of the latter is that a design that uses no nuclear materials cannot produce nuclear yield. Several constraints on the design are dictated by external considerations: the potential use, the space and mass available from the delivery vehicle, and the current availability of special materials. In addition, physical properties of materials and general policy concerning nuclear weapons impose other limits. The task of the designer is to create a device meeting all these criteria.

That heuristics play a role in satisfying constraints we have already mentioned. Yet heuristics are not enough. Because the constraints may be encountered in different phases of the design process, violation of a constraint may not be discovered until late in the process. The expert system must be able to recover from the

discovery of the violation. It does this by backtracking [1]. In backtracking, once a violation is noted at any level of the design, the system withdraws its most recent choice and selects another alternative. If there are no unexplored alternatives at this level, then it goes further back until an untried choice is found.

A rule-based expert system does this backtracking. A typical rule might be

IF the mass of the fissile material is too large
THEN the fissile mass has been reduced
DO execute the mass reduction rule set

Such a rule asserts that the procedure to reduce the mass has been carried out. This "fact" can be used to trigger other rules in the process, for example, to inform the system that a procedure option (here, reduction of the mass) has already been tried and other options should be chosen. In this way the form of the rule permits backtracking.

In practice such chronological backtracking, which changes the most recently tried choice, may expend a lot of effort altering irrelevant choices. Dependency-directed backtracking withdraws the choices that matter; this is far more efficient. Rules of the form above establish the dependencies needed for directed backtracking.

Beyond its role in backtracking, this form of rule serves a second purpose. The DO part of the rule allows change of the parameters in a way that depends on the particular violation and that isolates specific calculations from the main control sequence. Because the DO can execute any Lisp function, some rules will have directly performed calculations by the time their conditions have been met. Others may run additional rule sets or separate "subexperts," as the example illustrates. Facts may be asserted to control the path of execution by the main expert system. This communication between expert systems is like the blackboard scheme of HEARSAY-II [2]. By combining heuristics to guide choice with cooperating expert systems to solve subproblems and backtracking to resolve conflicts, we expect to cope with the issues of large solution spaces and tentative reasoning characteristic of design.

Finally, the apprentice must generate data to drive a numerical simulation. This is done using data-driven techniques similar to those described above for Procon. One important difference is that the apprentice must translate from the objects that form the design to the input parameters of the simulation code.

We are still coding the apprentice. We have interviewed our domain experts, prepared specifications for the code, and developed dataflow diagrams. We have found a close correspondence between "knowledge engineering" and well-known (but not widely enough used in this area) techniques of structured analysis and design. These techniques of software engineering [3, 4] have proved widely useful for building more traditional computer systems. In our work on the apprentice, we have been struck by their aptness not only for code development but also for guiding us in acquiring and codifying the knowledge of the domain experts.

References

[1] Patrick Henry Winston, *Artificial Intelligence* (second edition). Addison-Wesley, Reading, MA, 1984.

[2] *Handbook of Artificial Intelligence,* vol 1., pp. 343-348. Avron Barr and Edward A. Feigenbaum, eds. William Kaufmann, Inc., Los Altos, CA, 1981.

[3] James Martin and Carma McClure, *Diagramming Techniques for Analysts and Programmers.* Prentice-Hall, Englewood Cliffs, NJ, 1985.

[4] Roger S. Pressman, *Software Engineering: A Practitioner's Approach.* McGraw-Hill, New York, 1982.

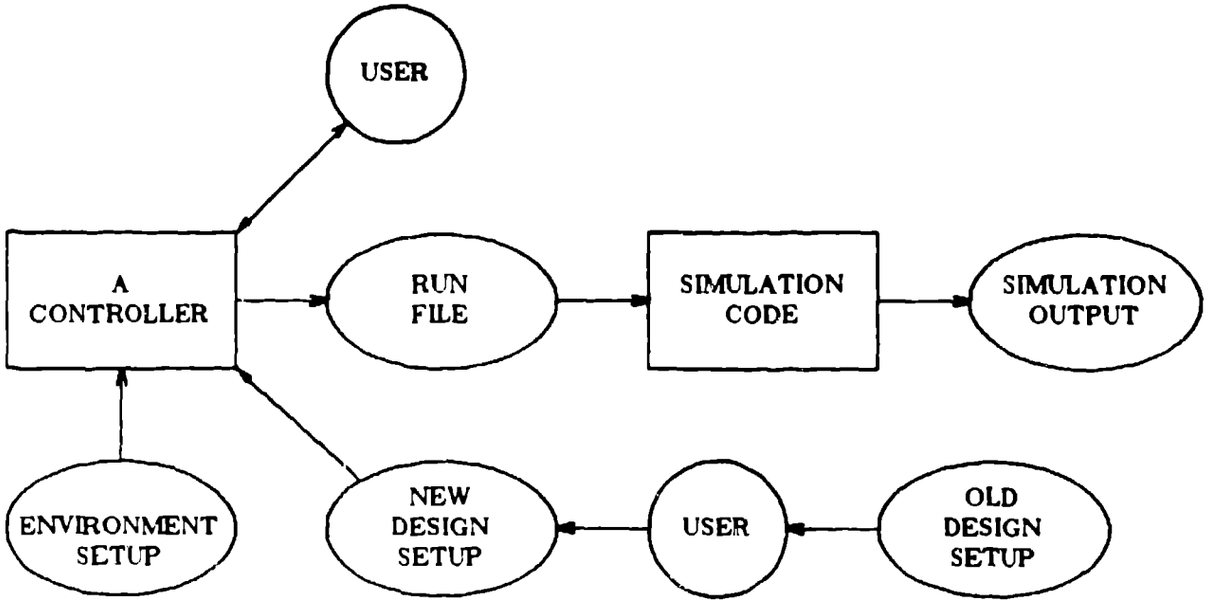


Fig. 1. A standard simulation process.
The designer is in the loop, modifying setups.
He must know internal details of the simulation codes.

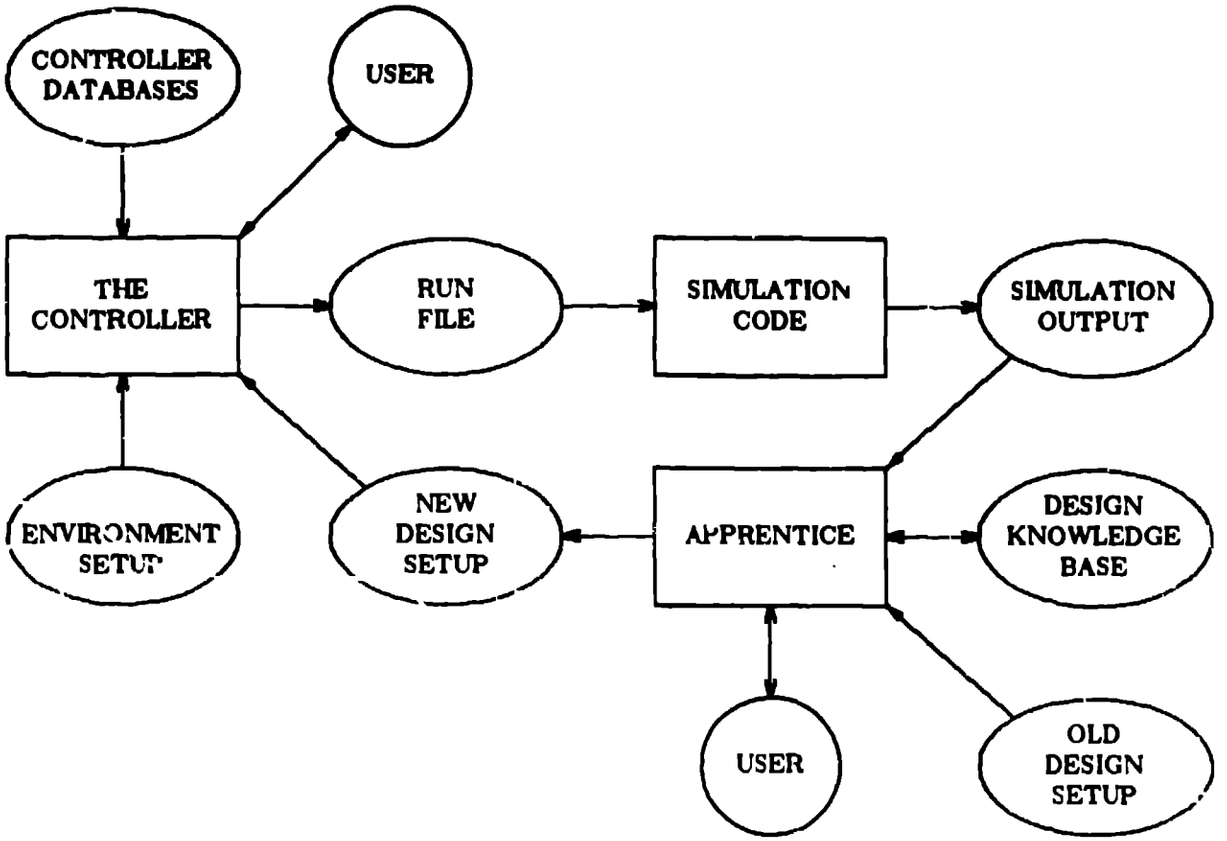


Fig 2. A simulation process with intelligent interfaces.
Expert systems can cope with internal details of the simulation codes.
Also they can help select alternative designs, track choices, and interpret results.

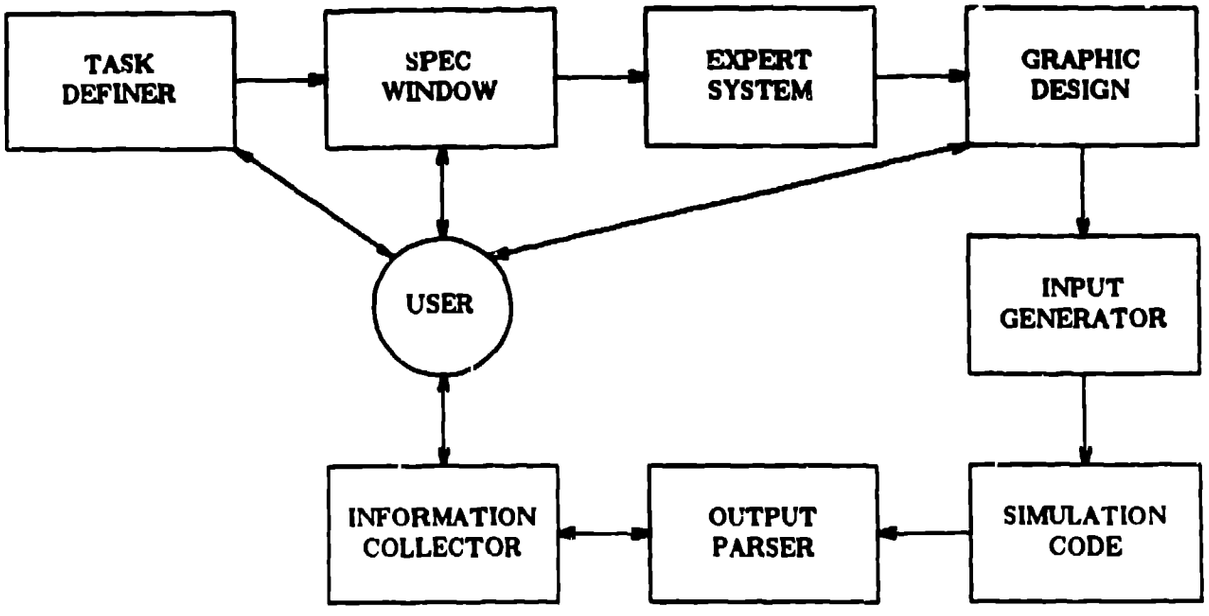


Fig. 3. The apprentice interacts with a designer.
It helps select a design, refine it, run the simulation, and interpret results.

EXPERT SYSTEMS FOR DESIGN AND SIMULATION

Jack Aldridge, John Cerutti,
Willard Draisin, and Michael Steuerwalt

Los Alamos National Laboratory

85/9/5

AIAA/NASA Symposium on Automation,
Robotics and Advanced Computing
for the National Space Program

EXPERT SYSTEMS CAN BE DEVELOPED TO SUPPORT DESIGN AND SIMULATION

Objectives
Procon to control simulation codes
Some practical difficulties
Apprentice to help design weapons
Summary

CAN AI TECHNOLOGY HELP?

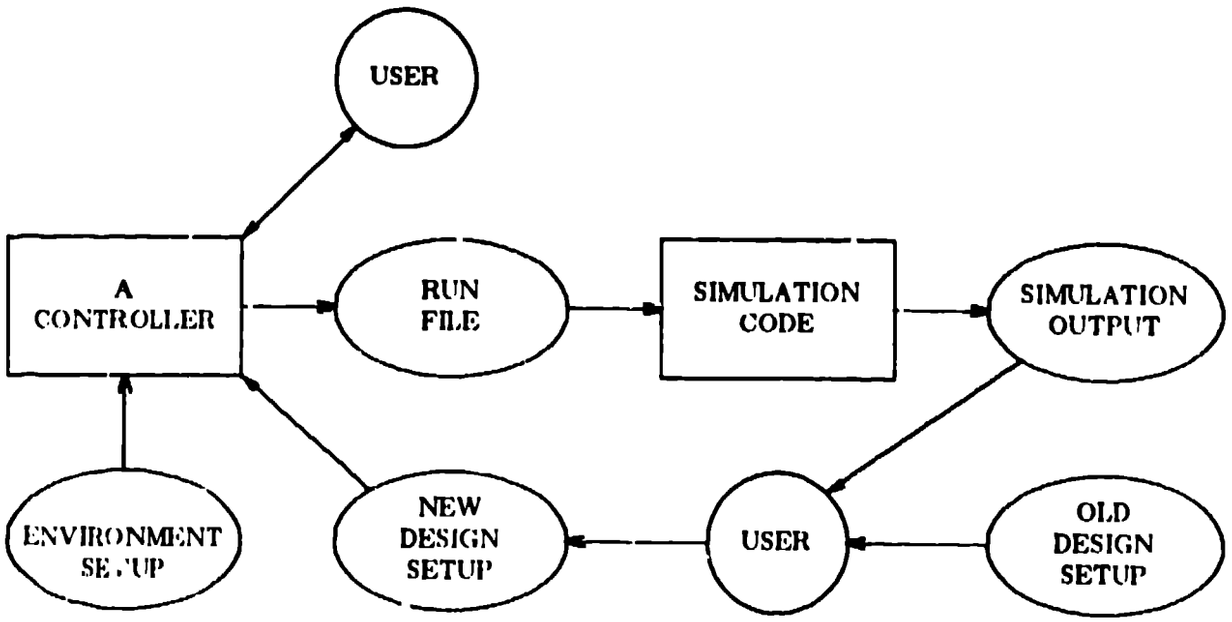
Do our present work better

- interpret between human expression and machine instruction
- enhance symbol-manipulation ability and blend with number-crunching

Do what is now impossible

STANDARD SIMULATION PUTS THE USER IN THE LOOP

He must know internal details of the simulation codes



BIG SIMULATION CODES ARE HARD TO UNDERSTAND

Simulation codes have

- 100,000-200,000 lines of code
- 500-2500 variables accessible to the user
- special capabilities for special circumstances

Documentation is usually inadequate

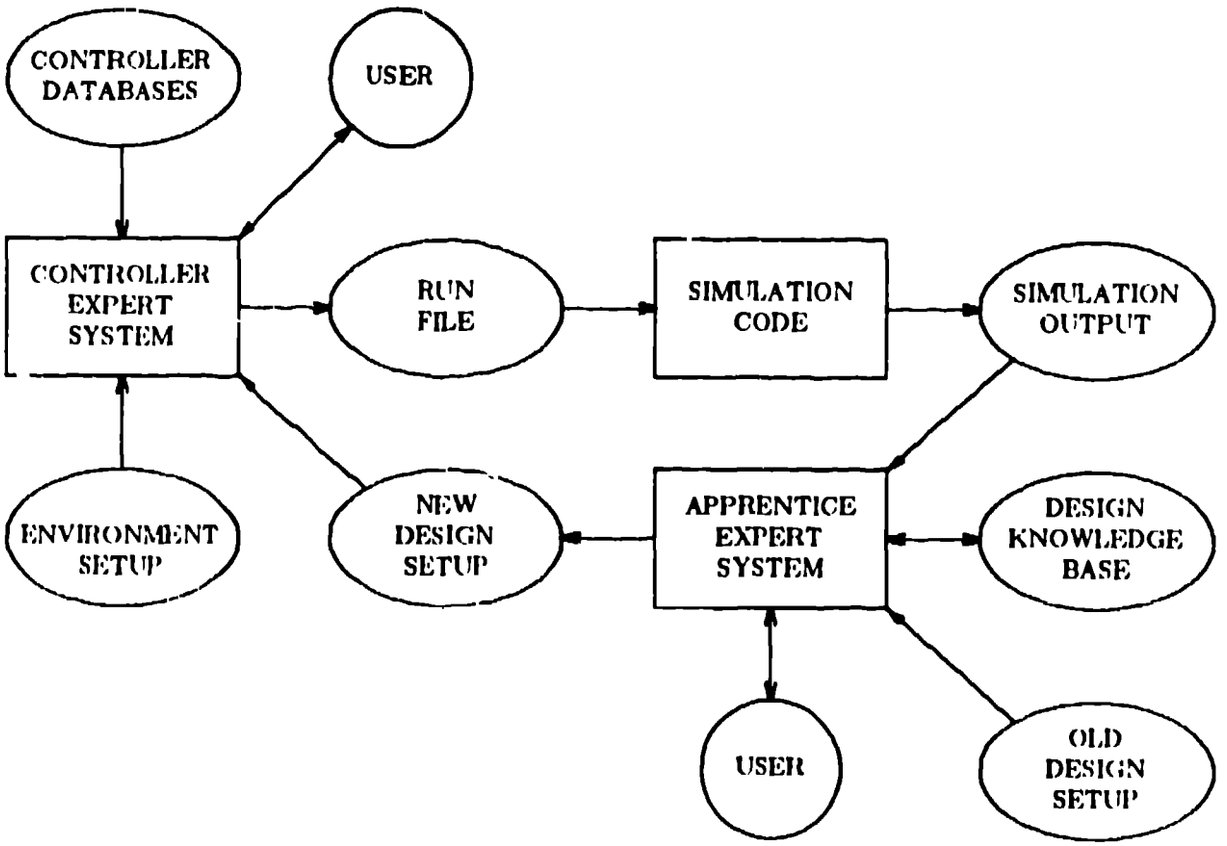
The code and its system environment keep changing

Users must communicate in the language of the code, not the language of the problem

SMART INTERFACES COPE WITH INTERNAL DETAILS

They can also help

- select alternative designs
- track choices
- interpret results



AI TECHNIQUES CAN HELP CONTROL SIMULATION CODES AND ASSIST IN DESIGN

Capture expertise
- reasoning aspects of design process
- knowledge of the internals of the simulation codes
- interpretation of results

Combine numeric and symbolic computation

Improve interfaces

WE ARE WORKING ON TOOLS TO HELP CONTROL BIG CODES AND SIMPLIFY DESIGN

Procon: help set up and submit production jobs
Apprentice: help design simple nuclear weapons

EXPERT SYSTEMS CAN BE DEVELOPED TO SUPPORT DESIGN AND SIMULATION

Objectives
Procon to control simulation codes
Some practical difficulties
Apprentice to help design weapons
Summary

PROCON SHOULD MAKE THE DESIGNER'S JOB EASIER

Help set up and submit "production" jobs
Stop jobs from aborting because of input errors
Be a uniform front end to many simulation codes

CONTROLLERS MONITOR AND MODIFY THE COURSE OF A PRODUCTION RUN

Production runs of a simulation can be expensive
- 2-100 hour runs, 5-50 files, 8-400 Mbytes

We rely on 3 expert programs as controllers

3 FTEs maintain the controllers

Changes to production codes or the operating environment require major modifications of the controllers

EXPERT PROGRAMS

- exhibit expert problem-solving behavior
- intertwine domain-specific knowledge with the code
- are hard to modify or use for a new problem
- are presently used to control simulation codes

EXPERT SYSTEMS

- exhibit expert problem-solving behavior
- comprise a base of information about a particular problem domain, together with an inference mechanism
- store domain knowledge (typically as rules) in a database separate from the executable code
- can easily be modified and extended, by altering the knowledge base
- become more expert in an incremental way

USE AI TECHNIQUES TO SOLVE THE PROBLEMS OF CONTROLLER EXPERT PROGRAMS

- build an expert system to control production runs
- control decisions and heuristics appear explicitly in a database
- easy, incremental extensions and modifications to the controller are possible
- one controller can provide a common user interface

WE BUILT A CONTROLLER SYSTEM THAT WORKS

Standard development cycle

- studied controllers used by designers
- picked a controller used by two design groups
- designed and implemented a first version on Vax/VMS

Procon has three main parts

- interface: creates an environment for the user
- checker: does some error checks, helps user build input describing the simulation
- generator: constructs job control stream and submits it for production

Cray version exists

PROCON HAS ADVANTAGES OVER STANDARD CONTROLLERS

Provides a common interface to several simulation codes

- fronts 2 simulation codes without rewriting executable code
- adding a third design code
- detects input errors in parameter names, values, subscripts, formats

Uses schema of isolating data from control

- easy to add new target codes, adapt to different environments

PROCON WILL BECOME MORE CAPABLE

Generator improvements
- fully data-driven; adapt to different production environments

Checker improvements
- check relations between input parameters
- check order of input

Database improvements
- full description of target codes
- provide aliasing of variables

Tutoring
- suggest to the user which simulation codes or which options to use

Aim at a uniform front end for design codes

EXPERT SYSTEMS CAN BE DEVELOPED TO SUPPORT DESIGN AND SIMULATION

Objectives
Procon to control simulation codes
Some practical difficulties
Apprentice to help design weapons
Summary

LIFE IS NOT A BED OF ROSES

Security contraints limit hardware, software, connections

Hardware is hard

- AI machines are expensive, single-user, with inadequate graphics or number-crunching power
- designers live on the Crays that run simulations
- high-resolution graphics on Tektronix 41xx
- high-speed communications between Crays, Vaxen, workstations

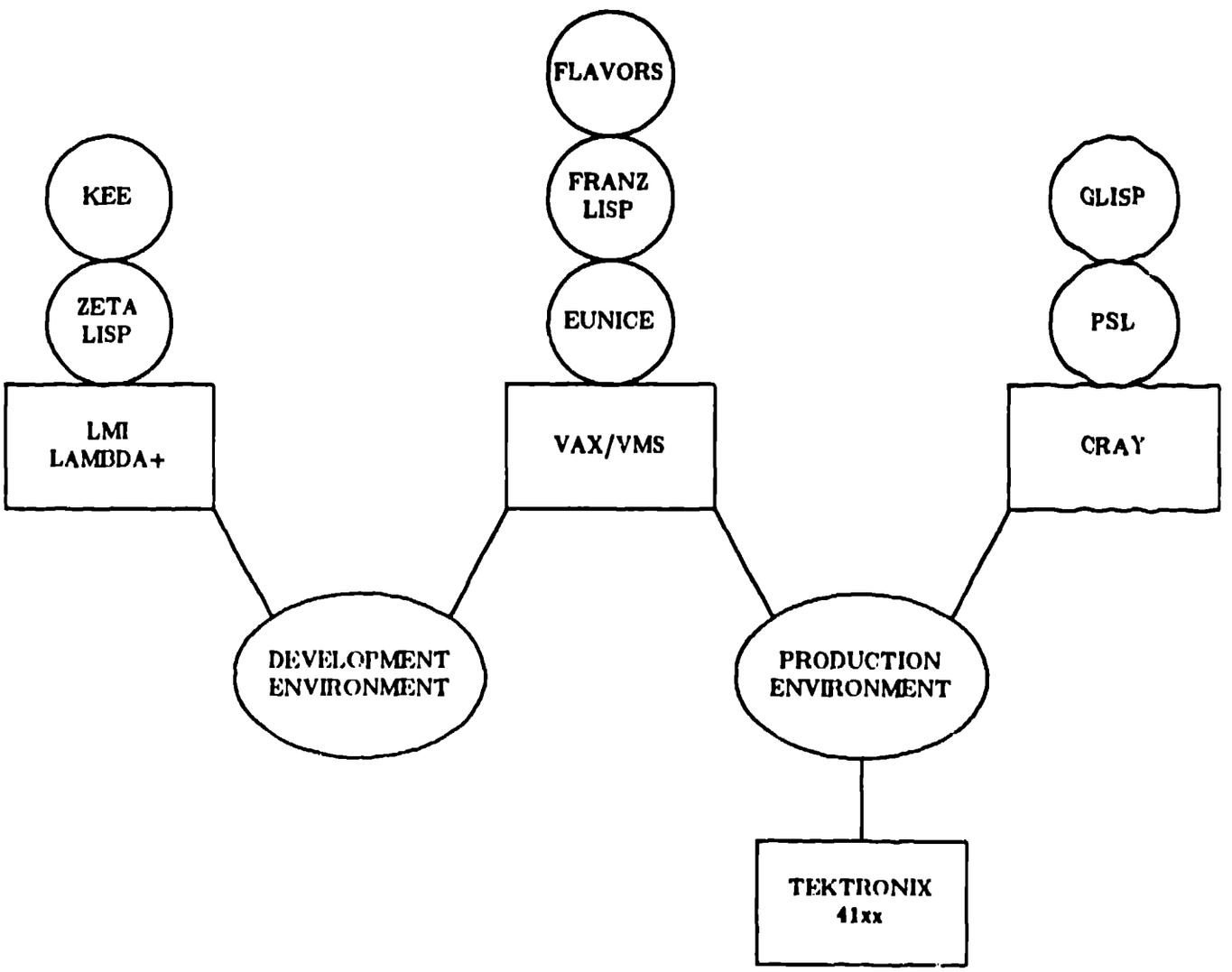
Software is also hard

- some Unix tools, no VMS tools when we started
- Eunice is inadequate
- many Lisps: PSL, Franz, Zeta, Inter, Common
- tools like GLisp, Flavors, KEE

Economics may be hardest

- high-res graphics terminals cost about $20K
- AI workstation + high-res color cost $90-160K

THE ENVIRONMENT IS COMPLICATED



GRAPHICS TERMINALS AND AI WORKSTATIONS FACILITATE DEVELOPMENT OF THE SYSTEMS

Graphics terminals give us the same interface to design codes

- same keyboard-display interaction
- explore ways to recover and display simulation results

AI workstations have the best development environment

- full Lisp environment: syntax-directed editor, debugger, windows
- KEE and other commercial products
- explore coupling between symbolic and numeric computation
- small machines can bring the apprentice to the designer

THE APPRENTICE SHOULD MAKE THE DESIGNER'S JOB EASIER

Turn descriptions into a design
Recall related design problems and refine the design
Call on standard production codes
Assist with parameter studies and interpret computational results
Incorporate special tools

Apply the power of the computer to the symbol-manipulation work that designers do

STUDY HOW WEAPONS ARE DESIGNED

- Identify the structure of the design process
- Define needed catalogs of design information
  - database of shots
  - nominal designs and their performances
  - characteristics of and relations between subsystems
- Uncover interplay between 1-D and 2-D simulations
- Pick class of weapons to try
- Outline design steps for the weapon

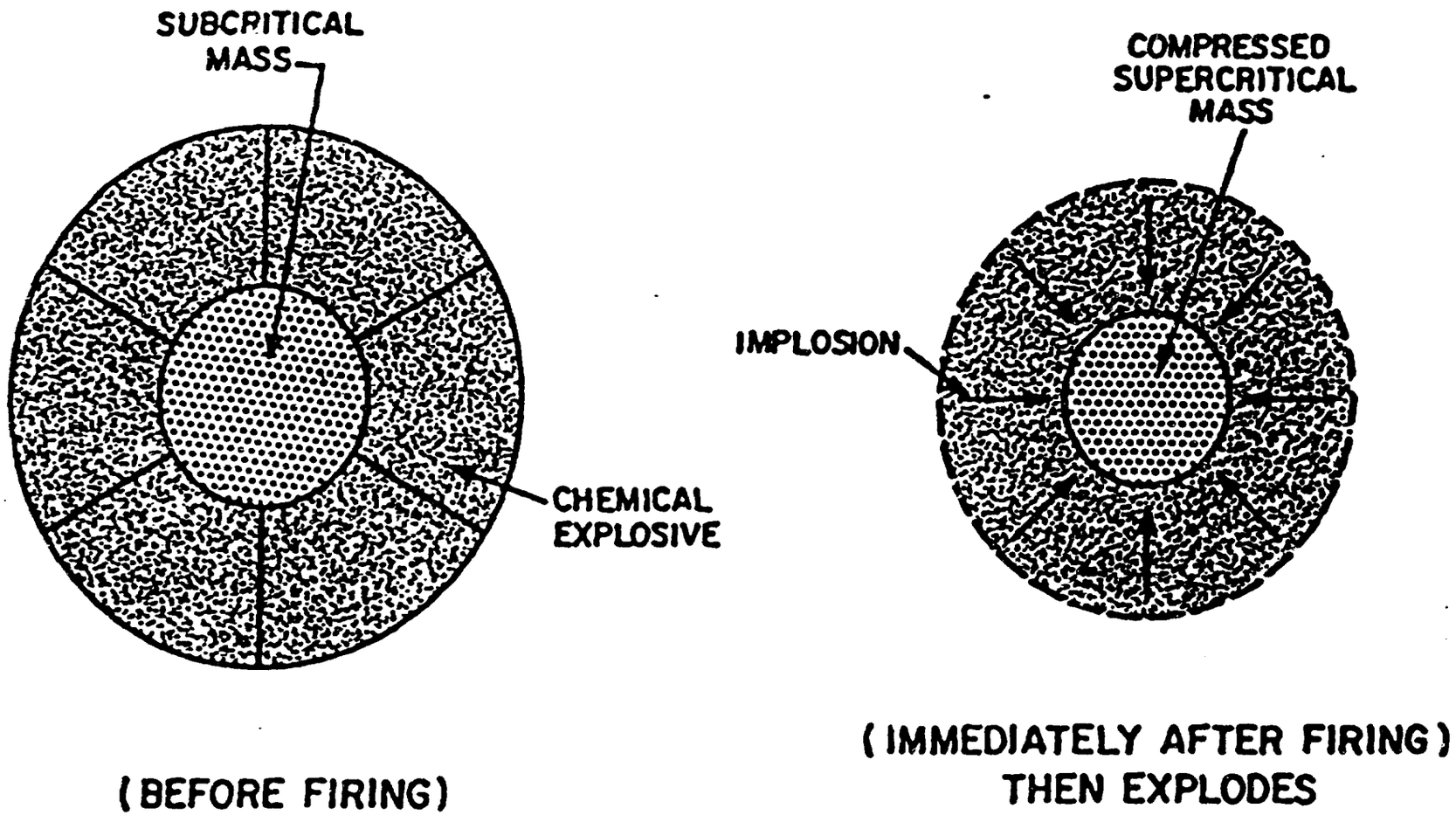


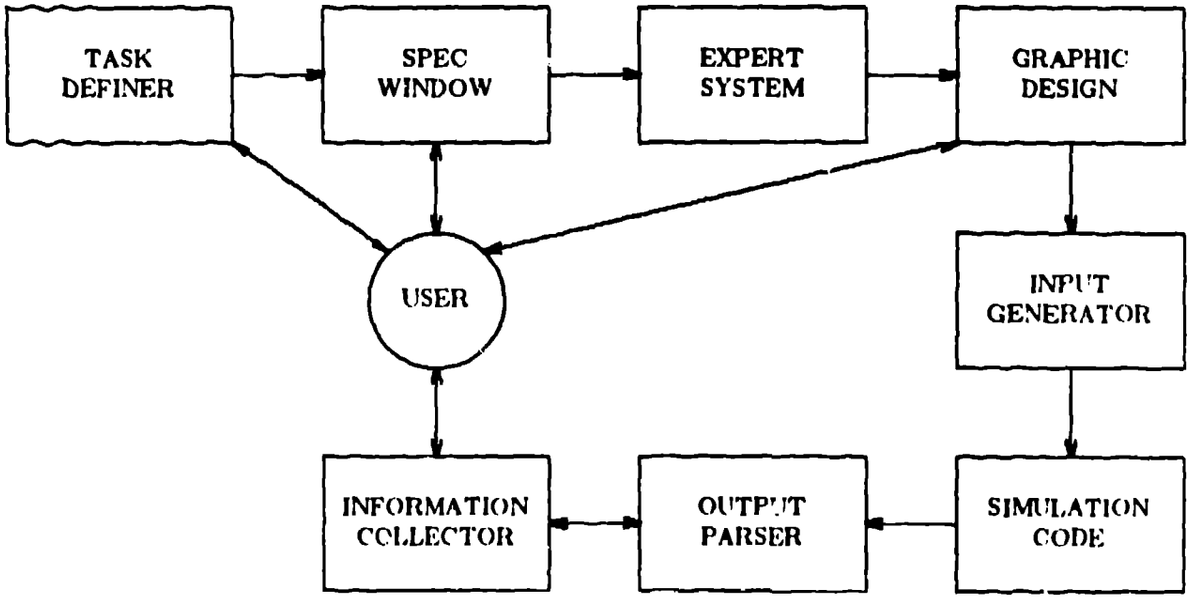
Figure 1.53. Principle of an implosion-type nuclear device.

A SIMPLE WEAPON IS AN EXCELLENT TESTBED FOR AN EXPERT SYSTEM

Simple enough to be doable
Uncovers problems that will arise with more complex weapons
Has been worked by an expert

THE APPRENTICE INTERACTS WITH A DESIGNER

- select a design
- refine it
- run the simulation code
- interpret results



THE APPRENTICE INTERACTS ...

Front end translates a description of a weapon into a design

Graphics box presents the design to the user, lets him manipulate it

Expert system helps refine the design

Setup-and-submit mechanism runs design codes

Collector recovers and displays results of design codes

THE APPRENTICE WILL GET SMARTER

Use special design codes to optimize designs
Address more complex weapons
Build a database of existing designs
Do parameter studies and interpret results
Run a sequence of several simulation codes

EXPERT SYSTEMS CAN BE DEVELOPED TO SUPPORT DESIGN AND SIMULATION

Objectives
Procon to control simulation codes
Some practical difficulties
Apprentice to help design weapons
Summary

USING AI TECHNIQUES CAN HELP THE DESIGN PROCESS

The techniques are being applied to two specific tools

AI suggests a framework for combining numerical and symbolic computation

The apprentice can capture design decisions, strategies, and heuristics used now

An expert system controller can reduce the costs associated with present production controllers